FAST QMC MATRIX-VECTOR MULTIPLICATION

JOSEF DICK, FRANCES Y. KUO, QUOC T. LE GIA, CHRISTOPH SCHWAB

Abstract. Quasi-Monte Carlo (QMC) rules $1/N \sum_{n=0}^{N-1} f(\boldsymbol{y}_n A)$ can be used to approximate integrals of the form $\int_{[0,1]^s} f(\boldsymbol{y}A)\,\mathrm{d}\boldsymbol{y}$, where A is a matrix and $\boldsymbol{y}$ is row vector. This type of integral arises for example from the simulation of a normal distribution with a general covariance matrix, from the approximation of the expectation value of solutions of PDEs with random coefficients, or from applications from statistics. In this paper we design QMC quadrature points $\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{N-1} \in [0,1]^s$ such that for the matrix $Y = (\boldsymbol{y}_0^\top, \ldots, \boldsymbol{y}_{N-1}^\top)^\top$ whose rows are the quadrature points, one can use the fast Fourier transform to compute the matrix-vector product $Y\boldsymbol{a}^\top$, $\boldsymbol{a} \in \mathbb{R}^s$, in $\mathcal{O}(N \log N)$ operations and at most $s-1$ extra additions. The proposed method can be applied to lattice rules, polynomial lattice rules and a certain type of Korobov p-set.

The approach is illustrated computationally by three numerical experiments. The first test considers the generation of points with normal distribution and general covariance matrix, the second test applies QMC to high-dimensional, affine-parametric, elliptic partial differential equations with uniformly distributed random coefficients, and the third test addresses Finite-Element discretizations of elliptic partial differential equations with high-dimensional, log-normal random input data. All numerical tests show a significant speed-up of the computation times of the fast QMC matrix method compared to a conventional implementation as the dimension becomes large.

Key words: Quasi-Monte Carlo, fast Fourier transform, lattice rule, polynomial lattice rule, Korobov p-set, high-dimensional integration, partial differential equations with random input.

MSC Class: 65C05

1. Introduction

We are interested in numerical approximations of integrals of the form

$$\int_U f(\boldsymbol{y}A)\,\mu(\mathrm{d}\boldsymbol{y}), \tag{1}$$

where the parameter domain U is a subset of $\mathbb{R}^s$, μ is a probability measure on U, $\boldsymbol{y}$ is a $1 \times s$ row vector, and A is an $s \times t$ real matrix. Often we have $t = s$, but there are also instances where t is much larger than s, see Section 3 below. We approximate these integrals by equal-weight quadrature rules

$$\frac{1}{N} \sum_{n=0}^{N-1} f(\boldsymbol{y}_n A), \tag{2}$$

where $\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{N-1} \in U$ are quadrature points which are expressed again as row vectors (using row vectors merely simplifies our notation later on, it is not a necessity). We are interested in cases where the computation of $\boldsymbol{y}_n A$ for $n = 0, \ldots, N-1$ is a significant factor in the computation of (2) and where N is significantly smaller than 2^s (say $N \approx s^\kappa$ for some $\kappa > 0$). The condition $N \ll 2^s$ is often naturally satisfied: for instance if the dimension s is very large, say $s > 100$, then the number of points N which can be used on current computers is much smaller than $2^{100} \approx 10^{30}$; another instance arises for example if the dimension is derived from a discretization or approximation scheme where one needs to increase the dimension s together with N in order to reduce the discretization or approximation error in a way such that $N \ll 2^s$. Examples where such situations arise naturally are given in Section 3.

Returning to the approximation of (1) by (2), one concrete example of our setting arises from taking the expectation of some quantity of interest with respect to the multivariate normal density with a general covariance matrix $\Sigma \in \mathbb{R}^{s \times s}$,

$$\mathbb{E}[f] = \int_{\mathbb{R}^s} f(\boldsymbol{z}) \frac{\exp(-\frac{1}{2}\boldsymbol{z}\Sigma^{-1}\boldsymbol{z}^{\mathsf{T}})}{\sqrt{(2\pi)^s \det(\Sigma)}} \,\mathrm{d}\boldsymbol{z}.$$

Using a factorization $\Sigma = A^{\mathsf{T}} A$ together with the substitution $\boldsymbol{z} = \boldsymbol{y}A$, we arrive at the integral (1), with $U = \mathbb{R}^s$, $s = t$, and with μ being the standard product Gaussian measure. (If the mean for the multivariate normal density is nonzero then a translation should be included in the substitution, but the general principle remains the same.) The method (2) can be interpreted as the simple *Monte Carlo* approximation with $\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{N-1} \in \mathbb{R}^s$ being i.i.d. standard Gaussian random vectors, and the computation of $\boldsymbol{y}_n A$ for $n = 0, \ldots, N-1$ can be interpreted as generating normally distributed points in $\mathbb{R}^s$ with the given covariance matrix Σ. The method (2) can also be interpreted as a *quasi-Monte Carlo* (QMC) approximation with $\boldsymbol{y}_n = \Phi^{-1}(\boldsymbol{x}_n)$ for $n = 0, \ldots, N-1$, where $\boldsymbol{x}_0, \cdots, \boldsymbol{x}_{N-1} \in [0,1]^s$ are deterministic QMC sample points, and where $\Phi^{-1} : [0,1] \to \mathbb{R}$ denotes the inverse of the standard normal distribution function and is applied component-wise to a vector. We will consider this example in Subsection 3.1.

Several papers studied how one can obtain matrices A in special circumstances which allow a fast matrix-vector multiplication. In the context of generating Brownian paths in mathematical finance, it is well known that the "standard construction" (corresponding to the Cholesky factorization of Σ) and the "Brownian Bridge construction" can be done in $\mathcal{O}(s)$ operations without explicitly carrying out the matrix-vector multiplication (see e.g., [11]), while the "principal components construction" (corresponding to the eigenvalue decomposition of Σ) can sometimes be carried out using Discrete Sine Transform in $\mathcal{O}(s \log s)$ operations (see e.g., [10, 22]); other fast orthogonal transform strategies can also be used (see e.g., [18]). In the context of PDEs with random coefficients, it is known that circulant embedding techniques can be applied in the generation of stationary Gaussian random fields so that Fast Fourier Transform (FFT) can be used (see e.g., [8, 13]).

The approach in this paper differs from all of the above in that *we do not try to modify the matrix A, but rather modify the quadrature points $\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{N-1}$ to reduce the cost of computing the N matrix-vector products $\boldsymbol{y}_n A$ for $n = 0, \ldots, N-1$. This implies that we do not require any structure in the matrix A, and so our approach is applicable in general circumstances.* For example, in some finance problems the payoff depends not only on the Brownian paths but also on a basket of assets; our approach can be used to speed up the matrix-vector multiplications with a factorization of the covariance matrix among the assets. Another example arises from the maximum likelihood estimation of generalised response models in statistics: the change of variables strategy proposed in [16] requires one to numerically compute the stationary point of the exponent in the likelihood integrand function and the corresponding quadratic term in the multivariate Taylor expansion; our approach can be used to speed up the matrix-vector multiplications with a factorization of this numerically computed Hessian matrix. Yet another important example arises from parametric PDEs on high-dimensional parameter spaces, which appear in computational uncertainty quantification; the presently proposed approach can be used for both the so-called "uniform" and "log-normal" inputs (see e.g., [12, 13, 17]). We will consider these PDE applications in Subsections 3.2 and 3.3.

To explain the idea behind our approach, we introduce the matrix

$$Y = \begin{pmatrix} \boldsymbol{y}_0 \\ \vdots \\ \boldsymbol{y}_{N-1} \end{pmatrix} \in \mathbb{R}^{N \times s},$$

and we want to have a fast method to compute

$$YA = B = \begin{pmatrix} \boldsymbol{b}_0 \\ \vdots \\ \boldsymbol{b}_{N-1} \end{pmatrix}.$$

To compute (2), we propose to first compute the product $B = YA$, store the matrix B, and then evaluate

$$\frac{1}{N} \sum_{n=0}^{N-1} f(\boldsymbol{b}_n).$$

Thus this method requires $\mathcal{O}(Nt)$ storage. In general, the computation of YA requires $\mathcal{O}(Ns\,t)$ operations, and the quadrature sum requires $\mathcal{O}(N)$ operations. In the following we construct quadrature points $\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{N-1}$ for which the matrix Y permits a matrix-vector multiplication $Y\boldsymbol{a}$ in $\mathcal{O}(N \log N)$ operations, where $\boldsymbol{a}$ can be any *column* of the matrix A. The computation of YA then reduces to $\mathcal{O}(t\,N \log N)$ operations, instead of $\mathcal{O}(Ns\,t)$ operations for the straight forward implementation. This leads to significant speedup provided that N is much smaller than 2^s.

The basic idea of the proposed approach is to find quadrature point sets $\{\boldsymbol{y}_0, \ldots, \boldsymbol{y}_{N-1}\} \in \mathbb{R}^s$ with a specific ordering such that the matrix

$$Y' = \begin{pmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_{N-1} \end{pmatrix} \in \mathbb{R}^{(N-1) \times s}$$

has a factorization of the form

$$Y' = ZP,$$

where $Z \in \mathbb{R}^{(N-1)\times(N-1)}$ is a circulant matrix and $P \in \{0,1\}^{(N-1)\times s}$ is a matrix in which each column has at most one value which is 1 and with the remaining entries being 0. The special structure means that, for a given column vector $\boldsymbol{a}$, the column vector $\boldsymbol{a}' = P\boldsymbol{a}$ can be obtained in at most $\mathcal{O}(N)$ operations, and the matrix-vector multiplication $Z\boldsymbol{a}'$ can be computed in $\mathcal{O}(N \log N)$ operations using FFT. On the other hand, the computation of $\boldsymbol{y}_0\boldsymbol{a}$ requires at most $s-1$ additions and 1 multiplication. The vector $\boldsymbol{y}_0$ is separated out because typical QMC methods would lead to $\boldsymbol{y}_0$ being a constant vector. If $\boldsymbol{y}_0 = (0, \ldots, 0)$, then no extra computation is necessary.

In Section 2 we consider two important classes of QMC point sets whose structure facilitates the use of the presently proposed acceleration:

- point sets derived from lattice rules,
- the union of all Korobov lattice point sets (which is one class of "Korobov p-sets").

The same strategy can be applied also to polynomial lattice rules where the modulus is a primitive polynomial over a finite field $\mathbb{F}_b$ of order b, and to the union of all Korobov polynomial lattice rules.

Note that lattice rules and polynomial lattice rules can yield a convergence rate close to $\mathcal{O}(N^{-1})$ for sufficiently smooth integrands, with the implied constant independent of the integration-dimension s under appropriate conditions on the integrand function and the underlying function space setting, see e.g., [3]. The union of Korobov lattice point sets on the other hand achieves a convergence rate of $\mathcal{O}(N^{-1/2})$ for a much larger class of functions, and dimension independent error bounds can be obtained with significantly weaker assumptions [2, 6]. Thus, when the integrand is not smooth enough for lattice rules and polynomial lattice rules, the union of Korobov lattice point sets can be a good substitute for the simple Monte Carlo method so that the fast computation approach of this paper can be exploited.

To illustrate our method and to investigate numerically for which parameter ranges the improvements in the computational cost are visible, we consider three applications in Section 3. In Subsection 3.1 we generate normally distributed points with a general covariance matrix. In Subsections 3.2 and 3.3 we consider PDEs with random coefficients in the uniform case and log-normal case, respectively. The numerical results in Section 4 show that our method is significantly faster whenever the dimension becomes large.

2. Fast QMC matrix-vector multiplication

We explain the fast method for lattice point sets and the union of all Korobov lattice point sets. The basic idea also applies to polynomial lattice point sets and the union of all Korobov polynomial lattice point sets.

2.1. **Fast matrix-vector multiplication for lattice point sets.**

Our approach is very similar to the method used in [19] for the fast component-by-component construction of the generating vector for lattice rules, however, we apply it now to the matrix vector multiplication $Y\boldsymbol{a}$ rather then the component-by-component construction. For simplicity, we confine the exposition to cases where the number of points is a prime. Based on the presently developed ideas, the general case can be handled analogously with the method from [20].

Let N be a prime number, let $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$, and let $\mathbb{Z}_N^* = \{1, 2, \ldots, N-1\}$.

A lattice point set with generator $(g_1, g_2, \ldots, g_s) \in (\mathbb{Z}_N^*)^s$ is of the form

$$\left(\left\{\frac{ng_1}{N}\right\}, \left\{\frac{ng_2}{N}\right\}, \ldots, \left\{\frac{ng_s}{N}\right\}\right) \quad \text{for } n = 0, 1, \ldots, N-1,$$

where for nonnegative real numbers x we denote by $\{x\} = x - \lfloor x \rfloor$ the fractional part of x.

Let β be a primitive element of the multiplicative group $\mathbb{Z}_N^*$, i.e., we have $\{\beta^k \bmod N : k = 1, 2, \ldots, N-1\} = \mathbb{Z}_N^*$. As is well-known $\beta^{N-1} \equiv \beta^0 \equiv 1 \bmod N$. Moreover, its multiplicative inverse $\beta^{-1} \in \mathbb{Z}_N^*$ is also a primitive element. We write each component of the generating vector $(g_1, g_2 \ldots, g_s)$ as

$$g_j \equiv \beta^{c_j - 1} \bmod N \quad \text{for some} \quad 1 \le c_j \le N-1.$$

Note that the fast component-by-component algorithm of [19] for constructing the generating vector computes the values c_j as a by-product, and hence no additional computation is needed to obtain the values c_j in this case.

Clearly, the ordering of the QMC points does not affect the quadrature sum. We now specify a particular (unconventional) ordering which allows fast matrix-vector multiplications. We define $\boldsymbol{x}_0 = (0, \ldots, 0)$, and for $n = 1, 2, \ldots, N-1$ we define

$$\begin{aligned}
\boldsymbol{x}_n &= \left(\left\{\frac{\beta^{-(n-1)} g_1}{N}\right\}, \left\{\frac{\beta^{-(n-1)} g_2}{N}\right\}, \ldots, \left\{\frac{\beta^{-(n-1)} g_s}{N}\right\}\right) \\
&= \left(\left\{\frac{\beta^{-(n-1)} \beta^{c_1-1}}{N}\right\}, \left\{\frac{\beta^{-(n-1)} \beta^{c_2-1}}{N}\right\}, \ldots, \left\{\frac{\beta^{-(n-1)} \beta^{c_s-1}}{N}\right\}\right) \\
&= \left(\left\{\frac{\beta^{c_1-n}}{N}\right\}, \left\{\frac{\beta^{c_2-n}}{N}\right\}, \ldots, \left\{\frac{\beta^{c_s-n}}{N}\right\}\right).
\end{aligned}$$

In essence, we have changed the ordering by substituting the conventional index n with $\beta^{-(n-1)}$ and replacing each generating vector component g_j by β^{c_j-1}.

The quadrature points we consider in (2) are now given by

$$\boldsymbol{y}_n = \varphi(\boldsymbol{x}_n)$$

$$= (\varphi(x_{n,1}), \varphi(x_{n,2}), \ldots, \varphi(x_{n,s})) \quad \text{for } n = 0, 1, \ldots, N-1,$$

where *we apply the same univariate transformation* $\varphi : [0,1] \to \mathbb{R}$ *to every component of every point.* One example for such a transformation is $\varphi(x) = \Phi^{-1}(x)$, the inverse of the cumulative normal distribution function; this maps the points from $[0,1]^s$ to $\mathbb{R}^s$ as we already discussed in the introduction. Another example is $\varphi(x) = 1 - |2x - 1|$, the tent transform; results for lattice rules usually apply to periodic functions, applying the tent transform yields similar results for non-periodic functions, see [4]. The case where $\varphi(x) = x$ is included as a special case.

We discuss now the multiplication of the matrix Y with a column vector $\boldsymbol{a} \in \mathbb{R}^s$. Since $\boldsymbol{y}_0 = (\varphi(0), \varphi(0), \ldots, \varphi(0))$ we have

$$\boldsymbol{y}_0 \boldsymbol{a} = \varphi(0) \sum_{j=1}^{s} a_j.$$

In particular, if φ is the identity mapping then $\boldsymbol{y}_0 \boldsymbol{a} = 0$. Thus the first component can be computed using at most $s-1$ additions and 1 multiplication. We consider now the remaining matrix

$$Y' = \begin{pmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_{N-1} \end{pmatrix}.$$

In the following we show that Y' can be written as a product of a circulant matrix Z and a matrix P in which $N-1$ entries are 1 and the remaining entries are 0.

Recall that β is a primitive element of $\mathbb{Z}_N^*$. For $k \in \mathbb{Z}$ let

$$z_k = \varphi\left(\left\{\frac{\beta^k}{N}\right\}\right).$$

Then we have $z_k = z_{k+\ell(N-1)}$ for all $\ell \in \mathbb{Z}$. Let

$$Z = \begin{pmatrix} z_0 & z_1 & z_2 & \ldots & z_{N-3} & z_{N-2} \\ z_{N-2} & z_0 & z_1 & \ddots & \ddots & z_{N-3} \\ z_{N-3} & z_{N-2} & z_0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ z_2 & \ddots & \ddots & \ddots & z_0 & z_1 \\ z_1 & z_2 & \ldots & \ldots & z_{N-2} & z_0 \end{pmatrix}.$$

We define the matrix $P = (p_{k,j})_{1\le k\le N-1, 1\le j\le s} \in \{0,1\}^{(N-1)\times s}$ by

$$p_{k,j} = \begin{cases} 1 & \text{if } k = c_j, \\ 0 & \text{otherwise.} \end{cases}$$

Each column of the matrix P contains exactly one element 1, with the remaining elements being 0. It is now elementary to check that

$$Y' = ZP. \tag{3}$$

Note that the matrix Z is exactly the same as the matrix used in the fast component-by-component algorithm of [19]. In effect, the matrix P specifies which columns of Z to select (namely, the c_1-th, the c_2-th, $\ldots$, and the c_s-th) to recover Y'.

Let $\boldsymbol{a} \in \mathbb{R}^s$ be any column vector. Then $\boldsymbol{a}' = P\boldsymbol{a}$ can be obtained in at most $\mathcal{O}(N)$ operations due to the special structure of P, and the matrix-vector multiplication $Z\boldsymbol{a}'$ can be computed in $\mathcal{O}(N \log N)$ operations using FFT (see [9]) since Z is circulant. Thus $Y'\boldsymbol{a}$ can be computed in $\mathcal{O}(N \log N)$ operations, and hence the matrix-vector multiplication $Y\boldsymbol{a}$ can be carried out using $\mathcal{O}(N \log N)$ operations plus at most $s-1$ additions.

We remark that the formula (3) can also be used to generate the matrix Y', i.e., to generate the quadrature points in a fast way. Further, if one wants to store the point set, i.e., matrix Y, one can simply store the primitive root β and the s numbers $c_1, \ldots, c_s$.

The case where N is not a prime number can be treated as in [20].

We finish this subsection with a simple example to illustrate the idea.

Example 1. *Let $s = 3$, $N = 7$, and $(g_1, g_2, g_3) = (1, 5, 3)$. A primitive root for $\mathbb{Z}_7^*$ is $\beta = 3$, with multiplicative inverse $\beta^{-1} = 5$. We have*

$$\begin{aligned} g_1 &= 1 = 3^{1-1} \bmod 7 &&\Longrightarrow && c_1 = 1, \\ g_2 &= 5 = 3^{6-1} \bmod 7 &&\Longrightarrow && c_2 = 6, \\ g_3 &= 3 = 3^{2-1} \bmod 7 &&\Longrightarrow && c_3 = 2. \end{aligned}$$

The conventional ordering of the points and the new ordering are

$$\begin{cases} (0,0,0), \\ (\frac{1}{7}, \frac{5}{7}, \frac{3}{7}), \\ (\frac{2}{7}, \frac{3}{7}, \frac{6}{7}), \\ (\frac{3}{7}, \frac{1}{7}, \frac{2}{7}), \\ (\frac{4}{7}, \frac{6}{7}, \frac{5}{7}), \\ (\frac{5}{7}, \frac{4}{7}, \frac{1}{7}), \\ (\frac{6}{7}, \frac{2}{7}, \frac{4}{7}), \end{cases} \quad \textit{versus} \quad \begin{cases} \boldsymbol{x}_0 = (0,0,0), \\ \boldsymbol{x}_1 = (\{\frac{3^{1-1}}{7}\}, \{\frac{3^{6-1}}{7}\}, \{\frac{3^{2-1}}{7}\}) = (\frac{1}{7}, \frac{5}{7}, \frac{3}{7}), \\ \boldsymbol{x}_2 = (\{\frac{3^{1-2}}{7}\}, \{\frac{3^{6-2}}{7}\}, \{\frac{3^{2-2}}{7}\}) = (\frac{5}{7}, \frac{4}{7}, \frac{1}{7}), \\ \boldsymbol{x}_3 = (\{\frac{3^{1-3}}{7}\}, \{\frac{3^{6-3}}{7}\}, \{\frac{3^{2-3}}{7}\}) = (\frac{4}{7}, \frac{6}{7}, \frac{5}{7}), \\ \boldsymbol{x}_4 = (\{\frac{3^{1-4}}{7}\}, \{\frac{3^{6-4}}{7}\}, \{\frac{3^{2-4}}{7}\}) = (\frac{6}{7}, \frac{2}{7}, \frac{4}{7}), \\ \boldsymbol{x}_5 = (\{\frac{3^{1-5}}{7}\}, \{\frac{3^{6-5}}{7}\}, \{\frac{3^{2-5}}{7}\}) = (\frac{2}{7}, \frac{3}{7}, \frac{6}{7}), \\ \boldsymbol{x}_6 = (\{\frac{3^{1-6}}{7}\}, \{\frac{3^{6-6}}{7}\}, \{\frac{3^{2-6}}{7}\}) = (\frac{3}{7}, \frac{1}{7}, \frac{2}{7}). \end{cases}$$

It is easy to see that indeed

$$\underbrace{\begin{pmatrix} \varphi(\boldsymbol{x}_1) \\ \varphi(\boldsymbol{x}_2) \\ \varphi(\boldsymbol{x}_3) \\ \varphi(\boldsymbol{x}_4) \\ \varphi(\boldsymbol{x}_5) \\ \varphi(\boldsymbol{x}_6) \end{pmatrix}}_{Y'} = \underbrace{\begin{pmatrix} \varphi(\frac{1}{7}) & \varphi(\frac{3}{7}) & \varphi(\frac{2}{7}) & \varphi(\frac{6}{7}) & \varphi(\frac{4}{7}) & \varphi(\frac{5}{7}) \\ \varphi(\frac{5}{7}) & \varphi(\frac{1}{7}) & \varphi(\frac{3}{7}) & \varphi(\frac{2}{7}) & \varphi(\frac{6}{7}) & \varphi(\frac{4}{7}) \\ \varphi(\frac{4}{7}) & \varphi(\frac{5}{7}) & \varphi(\frac{1}{7}) & \varphi(\frac{3}{7}) & \varphi(\frac{2}{7}) & \varphi(\frac{6}{7}) \\ \varphi(\frac{6}{7}) & \varphi(\frac{4}{7}) & \varphi(\frac{5}{7}) & \varphi(\frac{1}{7}) & \varphi(\frac{3}{7}) & \varphi(\frac{2}{7}) \\ \varphi(\frac{2}{7}) & \varphi(\frac{6}{7}) & \varphi(\frac{4}{7}) & \varphi(\frac{5}{7}) & \varphi(\frac{1}{7}) & \varphi(\frac{3}{7}) \\ \varphi(\frac{3}{7}) & \varphi(\frac{2}{7}) & \varphi(\frac{6}{7}) & \varphi(\frac{4}{7}) & \varphi(\frac{5}{7}) & \varphi(\frac{1}{7}) \end{pmatrix}}_{Z} \underbrace{\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}}_{P}.$$

The matrix P specifies that we select the first, the sixth, and the second columns of Z, as indicated by the values of c_1, c_2, c_3, to recover Y'.

Remark 1. *The method discussed above works in the same way for polynomial lattice rules over a finite field $\mathbb{F}_b$ of order b.*

Remark 2. *The method does* not *work when we apply general randomization techniques such as "shifting" for lattice rules or "scrambling" for polynomial lattice rules. This is because the corresponding transformation φ in the mapping $\boldsymbol{y}_n = \varphi(\boldsymbol{x}_n)$ fails to be the same mapping in all coordinate directions. If we were to restrict all random shifts to be of the form $\boldsymbol{\Delta} = (\Delta, \ldots, \Delta) \in [0,1]^s$ in the case of lattice rules then the method would work.*

Remark 3. *Higher order polynomial lattice rules, which have been introduced in* [5]*, also fit into the structure used in this subsection since they can be viewed as the first b^m points of a polynomial lattice point sets with $b^{m\alpha}$ points, where $\alpha \in \mathbb{N}$ denotes the smoothness. Here $\alpha = 1$ corresponds to the classical polynomial lattice rules. However, if we use the method from this paper, then the matrix vector multiplication uses the full $b^{m\alpha}$ points, which means the matrix vector multiplication requires $\mathcal{O}(b^{m\alpha} m\alpha)$ operations, instead of $\mathcal{O}(b^m s)$ operations for a straightforward implementation. Thus this method is only advantageous if $b^{m\alpha} m\alpha \ll b^m s$. For $\alpha \geq 2$ this implies that $b^m \ll s$, which usually does not hold.*

2.2. **The union of all Korobov lattice point sets.**

Hua and Wang [15, Section 4.3] studied the point set

$$\left(\left\{ \frac{ng^0}{K} \right\}, \left\{ \frac{ng^1}{K} \right\}, \ldots, \left\{ \frac{ng^{s-1}}{K} \right\} \right), \quad \text{for } n, g = 1, 2 \ldots, K-1,$$

where K is a prime number. The number of points is $N = (K-1)^2$. This is essentially the union of all Korobov lattice point sets. (Note that Hua and Wang also included the cases $n = 0$ or $g = 0$, or both $n = g = 0$, but these only yield the zero vector.)

This point set achieves only a rate of convergence of the weighted star-discrepancy of $\mathcal{O}(N^{-1/2+\delta})$ for any $\delta > 0$, however, the dependence on the dimension of the weighted star-discrepancy is better than what is known for lattice point sets or polynomial lattice point sets in some circumstances, see [6] for more details.

We now specify a particular ordering of the points to allow fast matrix-vector multiplications. Let β be a primitive element in $\mathbb{Z}_K^*$. As in Subsection 2.1, we replace the index n in the conventional ordering by $\beta^{-(n-1)}$, and similarly we replace the index g by $\beta^{(g-1)}$. That is, for $n, g = 1, 2, \ldots, K-1$, we define

$$\begin{aligned}\boldsymbol{x}_{n,g} &= \left(\left\{\frac{\beta^{-(n-1)}\beta^{0(g-1)}}{K}\right\}, \left\{\frac{\beta^{-(n-1)}\beta^{1(g-1)}}{K}\right\}, \left\{\frac{\beta^{-(n-1)}\beta^{2(g-1)}}{K}\right\}, \ldots, \right. \\ &\qquad \left. \left\{\frac{\beta^{-(n-1)}\beta^{(s-1)(g-1)}}{K}\right\}\right) \\ &= \left(\left\{\frac{\beta^{c_{g,1}-n}}{K}\right\}, \left\{\frac{\beta^{c_{g,2}-n}}{K}\right\}, \left\{\frac{\beta^{c_{g,3}-n}}{K}\right\}, \ldots, \left\{\frac{\beta^{c_{g,s}-n}}{K}\right\}\right),\end{aligned}$$

with

$$c_{g,j} = (j-1)(g-1) + 1 \mod (K-1) \qquad \text{for} \quad j = 1, \ldots, s.$$

We also define

$$\boldsymbol{y}_{n,g} = \varphi(\boldsymbol{x}_{n,g}).$$

Finally we define the matrix

$$Y' = \begin{pmatrix} Y_1' \\ Y_2' \\ \vdots \\ Y_{K-1}' \end{pmatrix}, \quad \text{with} \quad Y_g' = \begin{pmatrix} \boldsymbol{y}_{1,g} \\ \boldsymbol{y}_{2,g} \\ \vdots \\ \boldsymbol{y}_{K-1,g} \end{pmatrix} \text{ for } g = 1, \ldots, K-1. \tag{4}$$

For the matrices Y_g' we can apply the method from Subsection 2.1 to write it as $Y_g' = ZP_g$ using the values of $c_{g,1}, \ldots, c_{g,s}$ so that a matrix-vector multiplication can be computed in at most $\mathcal{O}(K \log K)$ operations. Thus one matrix-vector product for the matrix Y' can be evaluated in $\mathcal{O}(N \log N)$ operations.

Remark 4. *The same strategy can be applied to the union of all Korobov polynomial lattice point sets.*

3. Applications

3.1. **Generation of normally distributed points with general covariance matrix.** In many applications one requires realizations of random variables in $\mathbb{R}^s$ which are normally distributed $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ with mean $\boldsymbol{\mu} = (\mu_1, \mu_2, \ldots, \mu_s)$ and covariance matrix $\Sigma \in \mathbb{R}^{s\times s}$. An algorithm to generate such random variables is described for instance in [14, Section 11.1.6] and works the following way. Let $A \in \mathbb{R}^{s\times s}$ be such that $A^\top A = \Sigma$; for example, A can be the upper triangular matrix in the Cholesky decomposition of Σ. To generate a point $(Z_1, Z_2, \ldots, Z_s) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$, one generates i.i.d. standard normal random variables $Y_1, Y_2, \ldots, Y_s \sim \mathcal{N}(0, 1)$ with mean 0 and variance 1 and then computes $(Z_1, Z_2, \ldots, Z_s) = (Y_1, Y_2, \ldots, Y_s)A + (\mu_1, \mu_2, \ldots, \mu_s)$.

As we already outlined in the introduction, this procedure can be implemented in the following way using deterministic QMC point sets. Let $\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_{N-1} \in [0, 1]^s$ be a set of quadrature points as described in Section 2. Let Φ^{-1} be the inverse of the cumulative normal distribution function. Set

$$\boldsymbol{y}_n = \Phi^{-1}(\boldsymbol{x}_n)$$

and compute

$$\boldsymbol{z}_n = \boldsymbol{y}_n A + \boldsymbol{\mu}. \tag{5}$$

Note that we do not assume any structure in the matrix A. This is contrary to a number of scenarios in e.g., mathematical finance; see our discussion in the introduction.

3.2. Partial differential equations with “uniform” random coefficients.

The matrix-vector multiplication also arises in applications of QMC for approximating linear functionals of solutions of PDEs with random coefficients, see e.g., [17].

A prototypical class of countably parametric, elliptic boundary value problems reads as

$$-\nabla \cdot (\mathfrak{a}(\vec{x}, \boldsymbol{y})\nabla u(\vec{x}, \boldsymbol{y})) = g(\vec{x}), \qquad \vec{x} \in D \subset \mathbb{R}^d,\ \boldsymbol{y} \in [-\tfrac{1}{2}, \tfrac{1}{2}]^{\mathbb{N}}, \tag{6}$$

$$u(\vec{x}, \boldsymbol{y}) = 0, \qquad \vec{x} \in \partial D, \tag{7}$$

where $\vec{x}$ is a vector in the convex, physical domain $D \subseteq \mathbb{R}^d$, and $\boldsymbol{y} = (y_1, y_2, \ldots)$ is a parametric sequence in $[-\frac{1}{2}, \frac{1}{2}]^{\mathbb{N}}$, with y_j being uniformly distributed on $[-\frac{1}{2}, \frac{1}{2}]$. Here, we distinguish notationally between a vector $\vec{x}$ in the spatial domain D and a vector $\boldsymbol{y}$ in the parametric domain $[-\frac{1}{2}, \frac{1}{2}]^{\mathbb{N}}$. The coefficient $\mathfrak{a}(\vec{x}, \boldsymbol{y})$ depends on the parameter sequence $\boldsymbol{y}$ in an affine manner, ie.

$$\mathfrak{a}(\vec{x}, \boldsymbol{y}) = \psi_0(\vec{x}) + \sum_{j=1}^{\infty} y_j\, \psi_j(\vec{x}) \tag{8}$$

In (8), the sequence of functions $\psi_j(\vec{x})$ for $j \geq 1$ is assumed to decay as $j \to \infty$ such that $\sum_{j\geq 1} \|\psi_j\|^p_{L^\infty(D)} < \infty$ for some $0 < p \leq 1$ and that

$\sum_{j\geq 1} \|\nabla \psi_j\|_{L^\infty(D)} < \infty$. For more background, necessary assumptions and theoretical results see [17].

In [17] the aim is to approximate the expected value $\mathbb{E}(G(u))$ with respect to the random sequence $\boldsymbol{y}$ of a linear functional G of the solution u of the PDE. The algorithm truncates the infinite sum in (8) after s terms (i.e. set $y_j = 0$ for $j > s$), solves the truncated problem using a (piecewise linear) finite element method with M mesh points, and then approximates the s-dimensional integral using an N-point QMC rule. The resulting three sources of errors, namely, the truncation error, the finite element error, and the quadrature error, need to be balanced. For instance, in the case that $\sum_{j=1}^\infty \|\psi_j\|_{L_\infty(D)}^{2/3} < \infty$ and that $g, G \in L^2(D)$, [17, Theorem 8.1] yields for continuous, piecewise linear Finite Element discretizations of (6) – (8) in D on quasiuniform meshes that we should choose

$$N \asymp s \asymp M^{2/d}, \tag{9}$$

where $\asymp$ indicates that the terms should be of the same order. In general we have $N \asymp s^\kappa$ for some small $\kappa > 0$ (i.e., $N \ll 2^s$). Thus the fast method of this paper can be advantageous (see the numerical results in Section 4).

Let $\phi_1, \phi_2, \ldots, \phi_M$ be a basis for the finite element space V_M. For each $0 \leq n < N$, let $u_M^{(n)} = \sum_{k=1}^M \widehat{u}_k^{(n)} \phi_k$ be the finite element approximation of the solution u of the PDE given the parameter $\boldsymbol{y}_n \in [-\frac{1}{2}, \frac{1}{2}]^s$. Then for each $0 \leq n < N$ we solve the linear system

$$(\widehat{u}_1^{(n)}, \widehat{u}_2^{(n)}, \ldots, \widehat{u}_M^{(n)})\, B(\boldsymbol{y}_n) = (\widehat{g}_1, \widehat{g}_2, \ldots, \widehat{g}_M) \tag{10}$$

for $(\widehat{u}_1^{(n)}, \widehat{u}_2^{(n)}, \ldots, \widehat{u}_M^{(n)})$, where $\widehat{g}_k = \int_D g(\vec{x})\, \phi_k(\vec{x})\, \mathrm{d}\vec{x}$ for $k = 1, \ldots, M$, and where the symmetric stiffness matrix $B(\boldsymbol{y}_n) = (b_{n,k,\ell})_{k,\ell}$ depends on $\boldsymbol{y}_n$ and has entries

$$b_{n,k,\ell} = \int_D \mathfrak{a}(\vec{x}, \boldsymbol{y}_n)\, \nabla\phi_k(\vec{x}) \cdot \nabla\phi_\ell(\vec{x})\, \mathrm{d}\vec{x}, \qquad 1 \leq k, \ell \leq M. \tag{11}$$

Note that, with a slight abuse of notation, $\mathfrak{a}(\vec{x}, \boldsymbol{y}_n)$ is given by (8) but truncated to s terms. The expected value of the solution can then be approximated by

$$\overline{u}(\vec{x}) = \frac{1}{N} \sum_{n=0}^{N-1} \sum_{k=1}^{M-1} \widehat{u}_k^{(n)} \phi_k(\vec{x}). \tag{12}$$

Due to the linear structure in (8) for the uniform case, we can write

$$b_{n,k,\ell} = a_{0,k,\ell} + \sum_{j=1}^{s} y_{n,j}\, a_{j,k,\ell}, \qquad 0 \leq n < N, \quad 1 \leq k, \ell \leq M,$$

where $y_{n,j}$ denotes the jth component of the nth point $\boldsymbol{y}_n$, and

$$a_{j,k,\ell} = \int_D \psi_j(\vec{x}) \nabla\phi_k(\vec{x}) \cdot \nabla\phi_\ell(\vec{x}) \,\mathrm{d}\vec{x}, \qquad 1 \leq k, \ell \leq M, \quad j \geq 0.$$

In the standard approach, one defines the symmetric matrices $A_j = (a_{j,k,\ell})_{1\leq k,\ell\leq M}$ and sets

$$B(\boldsymbol{y}_n) = A_0 + \sum_{j=1}^{s} y_{n,j}\, A_j, \qquad 0 \leq n < N.$$

Note that A_j is usually sparse, with only $\mathcal{O}(M)$ nonzero entries in the same position, depending only on the relative supports of the basis functions ϕ_k, which are thus in particular independent of j. The cost for computing $B(\boldsymbol{y}_n)$ for all $n = 0, \ldots, N-1$ is therefore $\mathcal{O}(MNs)$ operations.

The fast QMC matrix-vector approach is implemented as follows: let $Y = (\boldsymbol{y}_n)_{0\leq n<N}$ be the matrix whose rows are the quadrature points of the QMC rule, and let $\boldsymbol{a}_{k,\ell} = (a_{1,k,\ell}, \ldots, a_{s,k,\ell})^\top$ and $\boldsymbol{b}_{k,\ell} = (b_{0,k,\ell}, \ldots, b_{N-1,k,\ell})^\top$. Then compute

$$\boldsymbol{b}_{k,\ell} = (a_{0,k,\ell}, \ldots, a_{0,k,\ell})^\top + Y\boldsymbol{a}_{k,\ell} \quad \text{for all } 1 \leq k, \ell \leq M, \tag{13}$$

where each matrix-vector multiplication $Y\boldsymbol{a}_{k,\ell}$ should be done using the approach of the previous section. Since only $\mathcal{O}(M)$ vectors $\boldsymbol{a}_{k,\ell}$ are nonzero, this approach for obtaining $B(\boldsymbol{y}_n)$ for all $n = 0, \ldots, N-1$ therefore requires only $\mathcal{O}(M\,N \log N)$ operations.

The improvement in the computational cost is that we replaced a factor of $\mathcal{O}(s)$ by $\mathcal{O}(\log N)$ (or $\mathcal{O}(\log s)$ when $N \asymp s$, see (9)). However, this method requires us to store all the vectors $\boldsymbol{b}_{k,\ell}$. Using the sparsity of the stiffness matrices which is of $\mathcal{O}(M)$, we require $\mathcal{O}(MN)$ storage.

3.3. **Partial differential equations with "log-normal" random coefficients.** We consider the PDE (6) again but now we assume that the random diffusion coefficient is given in parametric form by

$$\mathfrak{a}(\vec{x}, \boldsymbol{y}) = \exp\left(\psi_0(\vec{x}) + \sum_{j=1}^{\infty} y_j\, \psi_j(\vec{x})\right), \quad y_j \sim \text{i.i.d. } \mathcal{N}(0,1).$$

This formula arises from the assumption that the logarithm of the random coefficient $\mathfrak{a}(\vec{x}, \cdot)$ is a Gaussian random field in the domain D, which is parametrized in terms of principal components of its covariance operator by a Karhunen-Loève expansion. We shall refer to this case as the "log-normal" case.

We may proceed as in the previous subsection, following (10)–(12). However, unlike the uniform case where linearity can be exploited, the integral (11) for the log-normal case generally cannot be solved explicitly so that we need to use a quadrature rule to approximate (11). Let $\vec{x}_{1,k,\ell}, \vec{x}_{2,k,\ell}, \ldots, \vec{x}_{I,k,\ell} \in D$ denote the set of quadrature points and $w_{1,k,\ell}, w_{2,k,\ell}, \ldots, w_{I,k,\ell} \in \mathbb{R}$ denote the corresponding quadrature weights which are used to approximate (11),

$$b_{n,k,\ell} \approx \widehat{b}_{n,k,\ell} = \sum_{i=1}^{I} w_{i,k,\ell}\, \mathfrak{a}(\vec{x}_{i,k,\ell}, \boldsymbol{y}_n) \nabla \phi_k(\vec{x}_{i,k,\ell}) \cdot \nabla \phi_\ell(\vec{x}_{i,k,\ell}) \tag{14}$$

for $0 \le n < N$ and $1 \le k, \ell \le M$. Let $\widehat{B}(\boldsymbol{y}_n) = (\widehat{b}_{n,k,\ell})_{k,\ell}$. Thus we need to compute

(15)

$$\mathfrak{a}(\vec{x}_{i,k,\ell}, \boldsymbol{y}_n) = \exp(\theta_{i,k,\ell,n}), \quad \theta_{i,k,\ell,n} = \psi_0(\vec{x}_{i,k,\ell}) + \sum_{j=1}^{s} y_{n,j}\, \psi_j(\vec{x}_{i,k,\ell}),$$

for all $1 \le i \le I$, $0 \le n < N$ and $1 \le k, \ell \le M$ such that $\nabla \phi_k(\vec{x}_{i,k,\ell}) \cdot \nabla \phi_\ell(\vec{x}_{i,k,\ell})$ is nonzero. The number of these nonzero inner products is $\mathcal{O}(M)$ since for a fixed k, the number of ℓ such that the intersection of the supports of ϕ_k and ϕ_ℓ is nonempty does not depend on M. The standard approach to obtain $\widehat{B}(\boldsymbol{y}_n)$ for all $n = 0, \ldots, N-1$ therefore requires $\mathcal{O}(I\, M\, N\, s)$ operations.

We now describe the fast approach. Let

$$\Theta_{k,\ell} = (\theta_{i,k,\ell,n})_{\substack{1 \le i \le I \\ 0 \le n < N}},$$

$$\widehat{\Psi}_{i,k,\ell} = \begin{pmatrix} \psi_0(\vec{x}_{i,k,\ell}) \\ \psi_0(\vec{x}_{i,k,\ell}) \\ \vdots \\ \psi_0(\vec{x}_{i,k,\ell}) \end{pmatrix}, \quad \widehat{\Psi}_{k,\ell} = (\widehat{\Psi}_{1,k,\ell}, \ldots, \widehat{\Psi}_{I,k,\ell}) \in \mathbb{R}^{N \times I},$$

$$\Psi_{i,k,\ell} = \begin{pmatrix} \psi_1(\vec{x}_{i,k,\ell}) \\ \psi_2(\vec{x}_{i,k,\ell}) \\ \vdots \\ \psi_s(\vec{x}_{i,k,\ell}) \end{pmatrix}, \quad \Psi_{k,\ell} = (\Psi_{1,k,\ell}, \ldots, \Psi_{I,k,\ell}) \in \mathbb{R}^{s \times I}.$$

Then (15) can be written in matrix form as

$$\Theta_{k,\ell} = \widehat{\Psi}_{k,\ell} + Y \Psi_{k,\ell}, \tag{16}$$

where the multiplication $Y\Psi_{k,\ell}$ should be done as described in Section 2. Hence (15) can be computed using the fast QMC matrix method and $\widehat{B}(\boldsymbol{y}_n)$ for all $0 \le n < N$ can be computed in $\mathcal{O}(I\, M\, N \log N)$ operations. Again, the saving is that we replaced the factor $\mathcal{O}(s)$ by a factor $\mathcal{O}(\log N)$ in the computational cost.

4. Numerical experiments

In this section we carry out numerical experiments for the three applications from the previous section. In all the numerical experiments in the paper the times are averaged from 5 independent runs using Matlab R2013b on an Intel®Core™ Xeon E5-2650v2 CPU @ 2.6GHz.

Experiment 1: normally distributed points. We are interested in comparing the computation times using the standard approach of multiplying $\boldsymbol{y}_n A$ for $n = 0, 1, \ldots, N-1$, and the fast QMC matrix approach described in Subsection 2.1 with lattice point sets.

Table 1 shows the computation times in seconds for various values of N and s. The value on top shows the standard approach, whereas the value below shows the fast QMC matrix approach. For our experiments we chose $\boldsymbol{\mu} = (0, 0, \ldots, 0)$, and A a random upper triangular matrix with positive diagonal entries (so that A corresponds to the Cholesky factor of a random matrix Σ). The computation times do not include the component-by-component construction of the lattice generating vectors, the computation of $c_1, \ldots, c_s$ (since this information can be obtained from the fast component-by-component construction), nor the computation of $\Phi^{-1}(n/N)$ for $n = 0, 1, \ldots, N-1$ (since this computation is the same for both methods).

The numerical experiments in Table 1 show that there is an advantage using the fast QMC matrix-vector product if the dimension is large and the advantage grows as the dimension increases. This is in agreement with the theory since the computational cost in the standard approach is of order $\mathcal{O}(Ns^2)$ operations, whereas in the fast QMC matrix-vector approach it is of order $\mathcal{O}(s\, N \log N)$ operations. Recall that the fast QMC matrix method incurs a storage cost of $\mathcal{O}(Ns)$.

Experiment 2: the uniform case. We consider the ODE

$$-\frac{\mathrm{d}}{\mathrm{d}x}\left(a(x,\boldsymbol{y})\frac{\mathrm{d}}{\mathrm{d}x}u(x,\boldsymbol{y})\right) = g(x) \quad \text{for } x \in (0,1) \text{ and } \boldsymbol{y} \in [-\tfrac{1}{2}, \tfrac{1}{2}]^{\mathbb{N}}, \tag{17}$$

$$u(x,\boldsymbol{y}) = 0 \quad \text{for } x = 0, 1,$$

$$a(x,\boldsymbol{y}) = 2 + \sum_{j=1}^{\infty} y_j\, j^{-3/2} \sin(2\pi j x).$$

Thus $\sum_{j\geq 1} \|\psi_j\|_{L^\infty(0,1)}^{2/3+\varepsilon} < \infty$ for any $\varepsilon > 0$, and (9) implies that we should choose $N \asymp s$. In our experiments we choose $M = N = s$.

To obtain an approximation of the solution we use finite elements. Let $x_k = k/M$ for $k = 0, 1, \ldots, M$ and for $k = 1, 2, \ldots, M-1$ define

Method	N	$s=200$	$s=400$	$s=600$	$s=800$	$s=1000$
std.	16001	0.309	0.741	1.296	1.617	2.154
fast		0.164	0.301	0.450	0.589	0.741
std.	32003	0.589	1.468	2.435	3.063	4.238
fast		0.603	1.198	1.792	2.395	2.994
std.	64007	1.167	2.970	4.921	6.001	8.349
fast		1.804	3.853	5.551	7.582	9.827
std.	127997	2.579	5.889	9.490	11.891	16.818
fast		2.331	4.661	7.321	9.984	12.284
std.	256019	4.279	11.105	17.646	23.115	33.541
fast		5.401	10.933	16.174	24.147	26.898
std.	512009	8.885	23.368	31.942	48.059	66.378
fast		10.947	22.066	35.543	45.164	56.190

TABLE 1. Times (in seconds) to generate normally distributed points with random covariance matrix. The top row is the time required by using the standard approach, whereas the bottom row shows the time required using the fast QMC matrix-vector approach.

the hat function

$$\phi_k(x) = \begin{cases} (x - x_{k-1})M & \text{if } x_{k-1} \le x \le x_k, \\ (x_{k+1} - x)M & \text{if } x_k \le x \le x_{k+1}, \\ 0 & \text{otherwise.} \end{cases} \tag{18}$$

Then

$$a_{0,k,\ell} = \begin{cases} 4M & \text{if } k = \ell, \\ -2M & \text{if } |k-\ell| = 1, \\ 0 & \text{otherwise,} \end{cases}$$

and for $j \ge 1$ we have

$$\begin{aligned} a_{j,k,\ell} &= \int_0^1 j^{-3/2} \sin(2\pi j x) \phi_k'(x) \phi_\ell'(x) \, \mathrm{d}x \\ &= \begin{cases} \frac{M^2}{\pi j^{5/2}} \sin\left(\frac{2\pi j}{M}\right) \sin\left(\frac{2\pi j k}{M}\right) & \text{if } k = \ell, \\ -\frac{M^2}{\pi j^{5/2}} \sin\left(\frac{\pi j}{M}\right) \sin\left(\frac{\pi j(2k-1)}{M}\right) & \text{if } \ell = k-1, \\ -\frac{M^2}{\pi j^{5/2}} \sin\left(\frac{\pi j}{M}\right) \sin\left(\frac{\pi j(2k+1)}{M}\right) & \text{if } \ell = k+1, \\ 0 & \text{otherwise.} \end{cases} \end{aligned}$$

$M = s = 2N$									
N	67	127	257	509	1021	2053	4001	8009	16001
std.	1	5	31	190	1346	10610	74550	$\approx$144h	$\approx$1000h
fast	0.035	0.042	0.114	0.462	1.562	5.591	19.678	87.246	342.615

$M = s = \lceil\sqrt{N}\rceil$									
N	67	127	257	509	1021	2053	4001	8009	16001
std.	0.066	0.164	0.474	1.272	3.570	10.813	30.127	89.42	273.873
fast	0.012	0.015	0.028	0.059	0.126	0.265	0.516	1.113	2.443

$s = N$ and $M = N^2$				
N	67	127	257	509
std.	6	82	1699	27935
fast	0.243	1.385	11.268	107.042

Table 2. Times (in seconds) to obtain the average value of the finite element coefficients of the approximation (12) to (17). Top: $M = s = 2N$. Middle: $M = s = \lceil\sqrt{N}\rceil$. Bottom: $s = N$ and $M = N^2$.

Thus the matrices A_j and $B(\boldsymbol{y}_n)$ are tridiagonal. For simplicity we choose g such that $(\widehat{g}_1, \widehat{g}_2, \ldots, \widehat{g}_M) = (1, 1, \ldots, 1)$.

Table 2 shows the computation times comparing the standard approach with the fast QMC matrix method based on lattice point sets as described in Section 2.1. In this case the mapping in $\boldsymbol{y}_n = \varphi(\boldsymbol{x}_n)$ is $\varphi(x) = x - 1/2$, since the lattice points need to be translated from the usual unit cube $[0,1]^s$ to $[-\frac{1}{2}, \frac{1}{2}]^s$. Note that we do not apply any random shifting as analyzed in [17]. Since the dimension s is large, the fast QMC matrix method is very effective in reducing the computation times. Note that in Table 2 for the case $M = s = 2N$ the times for the standard method for $N = 8009$ and $N = 16001$ are in hours and are estimated from extrapolating on previous values in the table. The experiments show there is a clear advantage of fast QMC matrix-vector approach especially for large values of M, N and s.

Experiment 3: the log-normal case. In one space dimension, we consider the two-point boundary value problem for the parametric,

$M = s = 2N$									
N	67	127	257	509	1021	2053	4001	8009	16001
std.	0.028	0.051	0.140	0.436	1.734	15.173	84.381	614.636	4391.2
fast	0.040	0.033	0.094	0.326	1.122	4.296	15.203	60.546	270.691

$M = s = \lceil\sqrt{N}\rceil$									
N	67	127	257	509	1021	2053	4001	8009	16001
std.	0.030	0.053	0.090	0.182	0.375	0.791	1.609	4.100	7.874
fast	0.132	0.036	0.052	0.106	0.228	0.480	0.940	2.670	4.597

$s = N$ and $M = N^2$					
N	67	127	257	509	1021
std.	0.162	0.945	9.935	84.790	891.175
fast	0.204	1.084	10.154	83.861	746.907

Table 3. Times (in seconds) to obtain the average value of the finite element coefficients of the approximation (12) to (19). Top: $M = s = 2N$. Middle: $M = s = \lceil\sqrt{N}\rceil$. Bottom: $s = N$ and $M = N^2$.

second order ODE
(19)

$$\begin{aligned} -\frac{\mathrm{d}}{\mathrm{d}x}\left(a(x,\boldsymbol{y})\frac{\mathrm{d}}{\mathrm{d}x}u(x,\boldsymbol{y})\right) &= g(x) \quad \text{for } x \in (0,1) \text{ and } \boldsymbol{y} \in [-\tfrac{1}{2},\tfrac{1}{2}]^{\mathbb{N}}, \\ u(x,\boldsymbol{y}) &= 0 \quad \text{for } x = 0,1, \\ a(x,\boldsymbol{y}) &= \exp\left(2 + \sum_{j=1}^{\infty} y_j\, j^{-3/2}\sin(2\pi j x)\right). \end{aligned}$$

We use M finite elements to construct the approximate solutions as in (18). To compute (14), we use an equal weight quadrature with M (so $I = M$) points.

Table 3 shows the computation time for the log-normal case with different choices of number of finite elements M, the number of QMC points N and the truncated dimension s. As one would expect from the theory, the most significant advantage of the fast QMC matrix method occurs when 2^s is large compared to N, which is also reflected in the numerical results.

Acknowledgment

Josef Dick is the recipient of an Australian Research Council Queen Elizabeth II Fellowship (project number DP1097023). Frances Y. Kuo is the recipient of an Australian Research Council Future Fellowship (project number FT130100655). Quoc T. Le Gia was supported partially by the ARC Discovery Grant DP120101816. This research was supported under Australian Research Council's Discovery Projects funding scheme (project number DP150101770). Christoph Schwab acknowledges support through ERC and SNF.

References

[1] Y. Achdou and O. Pironneau, Computational methods for option pricing. Frontiers in Applied Mathematics, 30. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2005.

[2] J. Dick, Numerical integration of Hölder continuous, absolutely convergent Fourier, Fourier cosine, and Walsh series. J. Approx. Theory 183, 14–30 (2014).

[3] J. Dick, F. Y. Kuo, and I. H. Sloan, High dimensional integration – the quasi-Monte Carlo way. Acta Numer. 22, 133–288 (2013).

[4] J. Dick, D. Nuyens, and F. Pillichshammer, Lattice rules for non-periodic smooth integrands. Numer. Math. 126, 259–291 (2014).

[5] J. Dick and F. Pillichshammer, Strong tractability of multivariate integration of arbitrary high order using digitally shifted polynomial lattice rules. J. Complexity 23, 436–453 (2007).

[6] J. Dick and F. Pillichshammer, The weighted star-discrepancy of Korobov's p-sets. To appear in Proc. Amer. Math. Soc., 2015.

[7] J. Dick and F. Pillichshammer, Digital nets and sequences. Discrepancy theory and quasi-Monte Carlo integration. Cambridge University Press, Cambridge, 2010.

[8] C. R. Dietrich and G. H. Newsam, Fast and exact simulation of stationary Gaussian processes through circulant embedding of the covariance matrix. SIAM J. Sci. Comput. 18, 1088–1107 (1997).

[9] M. Frigo and S. G. Johnson, The design and implementation of FFTW3. Proceedings of the IEEE 93, 216–231 (2005).

[10] M. Giles, F. Y. Kuo, I. H. Sloan, and B. J. Waterhouse, Quasi-Monte Carlo for finance applications. ANZIAM Journal 50 (CTAC2008), C308 – C323 (2008).

[11] P. Glasserman, Monte Carlo methods in financial engineering. Applications of Mathematics (New York), 53. Stochastic Modelling and Applied Probability. Springer-Verlag, New York, 2004.

[12] I. G. Graham, F. Y. Kuo, J. A. Nichols, R. Scheichl, Ch. Schwab, and I. H. Sloan, Quasi-Monte Carlo finite element methods for elliptic pdes with lognormal random coefficients. To appear in Numerische Math., 2015.

[13] I. G. Graham, F. Y. Kuo, D. Nuyens, R. Scheichl, and I. H. Sloan, Quasi-Monte Carlo methods for elliptic PDEs with random coefficients and applications. J. Comput. Phys. 230, 3668–3694 (2011).

[14] W. Hörmann, J. Leydold, and G. Derflinger, Automatic nonuniform random variate generation. Springer, Berlin, 2004.

[15] L. K. Hua and Y. Wang, Applications of Number Theory to Numerical Analysis. Springer, Berlin, 1981.

[16] F. Y. Kuo, W. T. M. Dunsmuir, I. H. Sloan, M. P. Wand, and R. S. Womersley, Quasi-Monte Carlo for highly structured generalised response models. Methodol. Comput. Appl. Probab. 10, 239–275 (2008).

[17] F. Y. Kuo, Ch. Schwab, and I. H. Sloan, Quasi-Monte Carlo finite element methods for a class of elliptic partial differential equations with random coefficients. SIAM J. Numer. Anal. 50, 3351–3374 (2012).

[18] G. Leobacher, Fast orthogonal transforms and generation of Brownian paths. J. Complexity 28, 278–302 (2012).

[19] D. Nuyens and R. Cools, Fast algorithms for component-by-component construction of rank-1 lattice rules in shift-invariant reproducing kernel Hilbert spaces. Math. Comp. 75, 903–920 (2006).

[20] D. Nuyens and R. Cools, Fast component-by-component construction of rank-1 lattice rules with a non-prime number of points. J. Complexity 22, 4–28 (2006).

[21] D. Nuyens and R. Cools, Fast component-by-component construction, a reprise for different kernels. In: H. Niederreiter and D. Talay (eds.), Monte Carlo and quasi-Monte Carlo methods 2004, 373–387, Springer, Berlin, 2006.

[22] K. Scheicher, Complexity and effective dimension of discrete Lévy areas. J. Complexity 23, 152–168 (2007).

[23] Ch. Schwab, QMC Galerkin discretization of parametric operator equations. In: J. Dick, F. Y. Kuo, G. W. Peters and I. H. Sloan (eds.), Monte Carlo and quasi-Monte Carlo methods 2012, 613–629, Springer, Berlin 2013.

Addresses:

Josef Dick, Frances Y. Kuo, Quoc T. Le Gia, School of Mathematics and Statistics, The University of New South Wales, Sydney, 2052 NSW, Australia. e-mail: josef.dick, f.kuo, qlegia(AT)unsw.edu.au

Christoph Schwab, Seminar for Applied Mathematics, ETH, 8092 Zürich, Switzerland. e-mail: christoph.schwab(AT)sam.math.ethz.ch